

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

PATENT APPLICATION

**METHOD AND APPARATUS FOR FUNCTIONAL
REDUNDANCY CHECK MODE RECOVERY**

INVENTOR(S):

BRYANT E. BIGBEE OF SCOTSDALE, AZ
SHIVNANDAN KAUSHIK OF PORTLAND, OR
JAMES B. CROSSLAND OF BANKS, OR

PREPARED BY:

BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(512)330-0844

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EL863955828US

Date of Deposit: January 2, 2002

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to Box Patent Application, Commissioner of Patents and Trademarks, Washington, D.C. 20231

Dionne Robinson

(Typed or printed name of person mailing paper or fee)

(Signature of person mailing paper or fee)

Date

Dionne Robinson 01/02/02

20020102 08:00:00

METHOD AND APPARATUS FOR FUNCTIONAL REDUNDANCY CHECK MODE RECOVERY

Background of the Invention

[0001] The present invention relates generally to an improved method and apparatus for data processing. More particularly, the present invention relates to an improved method and apparatus for error detection and recovery within a data processing system. Still more particularly, the present invention relates to a method and apparatus for functional redundancy check mode recovery.

Description of the Related Art

[0002] Many modern data processing (e.g. computer) systems employ redundancy of system elements to improve system reliability and to decrease the amount of "downtime" resulting from component failures, faults, or internal errors. According to one technique for improving system reliability, two or more physical processing elements or "cores" are associated with and function as a single logical processor. Each processor core may have its own physical processor package or alternatively, two or more processor cores may be incorporated into a single processor package. The processor cores are operated in a functional redundancy check (FRC) mode in which identical instructions are provided to each core and concurrently executed on identical data so that any error occurring within a single core will produce an inconsistent result as compared to the remaining cores associated with a given processor. Specialized FRC logic performs this comparison and generates a signal such as an interrupt or exception once an inconsistent result has been detected.

[0003] In some conventional data processing systems, proprietary system management software has been utilized to perform similar error recovery while in other systems, specialized firmware has been used. According to one technique, hardware is utilized to generate a system-transparent interrupt in response to detecting a mismatch in redundantly operated processor cores. Specialized firmware receives the hardware-generated interrupt and performs appropriate recovery mechanisms. One drawback to this technique is that most conventional operating systems have a maximum tolerable interrupt latency before system errors or failures occur which varies dependent on the workload and associated devices of a data processing system.

Another drawback associated with this technique is that since processor state rather than software context is typically saved it may be more difficult to validate a system's capability to recover from an error occurring at any point of the system's operation.

[0004] According to another technique, hardware is utilized to generate a specific system-visible interrupt in response to detecting a mismatch in redundantly operated processor cores. A proprietary operating system program receives the hardware-generated interrupt and performs appropriate recovery mechanisms. Since software context rather than processor state is typically saved, it may be easier to validate a system's ability to recover from errors utilizing this technique. Significant drawbacks associated with this technique however include that it burdens the operating system, limits the number and types of operating systems which may be utilized with a particular hardware platform, limits the ability of the platform to evolve, and prevents differentiation in terms of the underlying platform implementation.

Brief Description of the Drawings

[0005] The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings in which similar references are utilized to indicate similar elements and in which:

[0006] **Figure 1** illustrates an abstraction layer block diagram of a data processing system according to one embodiment of the present invention;

[0007] **Figure 2** illustrates an exemplary data processing system according to one embodiment of the present invention;

[0008] **Figure 3** illustrates an exemplary block diagram of processor package 204a of **Figure 2** according to one embodiment of the present invention;

[0009] **Figure 4a** illustrates a process flow diagram for a first portion of a method according to one embodiment of the present invention; and

[0010] **Figure 4b** illustrates a process flow diagram for a second portion of a method according to one embodiment of the present invention.

Detailed Description

[0011] A method and apparatus for functional redundancy check mode recovery are described herein. In the following detailed description, numerous specific details such as specific data processing system, processor package, and abstraction layer interface architectures are set forth in order to provide a more thorough understanding of the present invention. It should be evident however, that these and other specific details described need not be utilized to practice the present invention. In other circumstances, well-known structures, elements, or connections have been omitted, or have not been described in particular detail in order to avoid unnecessarily obscuring the present invention.

[0012] Embodiments of the present invention may include data processing hardware, firmware, software, and various processing operations further described herein. The methods, features, and process operations of the present invention may be provided utilizing executable instructions embodied within a machine-accessible medium according to the present invention.

[0013] A machine-accessible medium may include any mechanism that provides (i.e., stores and/or transmits) information in a form accessible by a machine (e.g., a data processing system or computer). For example, a machine-accessible medium includes but is not limited to: read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); or the like. The executable instructions may be utilized to cause a general or special purpose processing element, programmed with the instructions, to perform methods or processes of the present invention. Alternatively, methods, features and operations of the present invention may be performed utilizing specific hardware components that contain hard-wired logic, or by any combination of programmed data processing components and custom hardware components.

[0014] **Figure 1** illustrates an abstraction layer block diagram of a data processing system 100 according to one embodiment of the present invention. The illustrated data processing system 100 includes hardware 102, system management programs according to the present invention including firmware 104 and operating system 106, and application programs 108. Hardware 102 may comprise a wide variety of devices

including processing elements, chipsets, volatile and nonvolatile memories, mass storage devices, and input-output (I/O) devices accessed utilizing firmware 104.

[0015] Firmware 104 contains basic input-output system (BIOS) program code for data processing system 100 and it initializes, configures, and tests hardware 102 within data processing system 100 and loads some or all of an operating system 106, typically from storage, into memory in a process called initial program load (IPL). Firmware 104 provides an abstraction layer between higher (application 108 and operating system 106) layers of data processing system 200 and hardware 102 and, in one embodiment, handles machine check exceptions. Once loaded, operating system 106, which may include a kernel and various interfaces, interrupt handlers, resource managers, and device drivers, manages the various application programs 108 and hardware 102 of data processing system 100.

[0016] In one embodiment, system 100 comprises an advanced configuration and power interface (ACPI)-compliant data processing system. Accordingly, operating system 106 includes operating system-directed power management (OSPM) program code, and an ACPI device driver and firmware 104 comprises an ACPI BIOS including ACPI machine language (AML) program code describing what hardware devices are present within data processing system 100, as well as their configuration and interfaces, via ACPI control methods, objects, registers, and tables. The disclosed ACPI device driver within operating system 106 acts as an AML interpreter to interpret and execute AML program code.

[0017] When an event occurs within data processing system 100, firmware 104 or hardware 102 causes a bit to be set within an event status register, such as a general purpose event (GPE) status register, to indicate the occurrence of the event. If a corresponding bit is set within an event enable register, the occurrence of the event may be signaled to operating system 106 via the generation of a system-visible interrupt (e.g. a system control interrupt or "SCI"). The operating system 106 receives the system-visible interrupt, determines what event caused the interrupt, and then services the event by executing program code (e.g. control methods) corresponding to bits set in the event status register. Control methods may be written in an ACPI source language (ASL) and compiled into AML for inclusion with the system's BIOS. The operating system 106 executes a control method by retrieving its

202010121001

associated AML code from BIOS and then interpreting the retrieved code utilizing its ACPI device driver/AML interpreter.

[0018] In another embodiment, firmware 104 includes an extensible firmware interface (EFI), a system abstraction level (SAL), and a processor abstraction level (PAL). The PAL provides an abstraction of implementation-specific processor features while the SAL provides a platform abstraction, isolating the EFI and operating system 106 from implementation-specific platform differences. The EFI in turn provides an application programming interface (API) to operating system 106, allowing the operating system 106 to interact with hardware 102 via the system and processor abstraction levels. When a machine check exception occurs, it may be handled by hardware, firmware, system management (e.g. operating system) software, or not at all. For example, single bit data error correcting code (ECC) errors occurring in processor cache may be handled by processor hardware while multi-bit data ECC or parity errors occurring in processor cache may be handled by an associated PAL.

[0019] In one embodiment, hardware 102 includes two or more processor packages, where each physical processor package includes two or more physical processor cores operating lock step in FRC mode as a single logical processor from the point of view of operating system 106. Hardware 102 of the disclosed embodiment further includes FRC logic. FRC logic detects errors within individual processor cores of the described embodiment which causes hardware 102 to break lock step between at least two processor cores within a processor package, disabling FRC mode operation. Hardware 102 then disables any of the package's error-containing cores, placing them into a "zombie" state, and generates a recoverable machine check exception. The generated machine check exception in turn triggers the execution of a processor error handler within a PAL of one of the package's error-free cores.

[0020] Thereafter, the processor error handler causes a platform error handler within a SAL associated with the package to be executed which initiates a platform-independent device removal sequence for a processor associated with the processor package by generating a system control interrupt (SCI) to operating system 106. In one embodiment, the SCI is generated using an INT_OUT command capability

[0021] In another embodiment, the disclosed processor error handler comprises a PALE_CHECK firmware routine within a PAL of one of the package's error-free cores and the platform error handler comprises SALE_ENTRY and SAL_CHECK firmware routines within an associated SAL. PALE_CHECK saves minimal processor state, determines if the machine check exception's associated errors/events are processor-related, saves processor-related error information, attempts to correct processor-related errors where possible, and then passes control to SALE_ENTRY which branches to SAL_CHECK. SAL_CHECK in turn saves additional processor and platform error and state information, attempts to correct any platform hardware-specific errors, and then generates an SCI to operating system 106 as described.

[0023] Operating system 106 next utilizes ACPI control methods to virtually eject the processor from data processing system 100. In one embodiment of the invention, this has no effect on hardware 102 other than to communicate to the ACPI BIOS within firmware 104 that the processor's package is being taken offline and to ensure that it is reset or "re-initialized" to the state needed for it to be subsequently awakened or "activated" by operating system 106. In another embodiment, virtually ejecting the processor results in the physical removal of power from its associated processor package.

6

embodiments of the invention, this reset may comprise a “cold” reset or “RESET” operation or a “warm” or “INIT” reset operation. An associated SAL reinitializes the package and re-enables FRC mode lock step operation between two or more of the package’s associated processor cores in response to detecting the ACPI BIOS-initiated reset operation. In yet another embodiment, the package’s associated SAL initiates a platform-independent device insertion sequence for the processor package’s associated processor by generating a system control interrupt (SCI) to operating system 106 via the previously disclosed INT_OUT command capability.

[0025] Operating system 106 receives the firmware-generated SCI and utilizes ACPI control methods to determine that the interrupt was caused by a request for processor insertion. Operating system 106 responsively modifies its internal data structures to reflect that the package’s associated processor is being brought online and then wakes and configures the associated processor for use. While specific firmware components have been described herein, it should be appreciated that alternative embodiments of the present invention may be implemented utilizing alternative firmware components and/or a combination of platform independent and dependent software components.

[0026] **Figure 2** illustrates an exemplary data processing system 200 according to one embodiment of the present invention. The data processing system 200 of the illustrated embodiment includes a first node 202a “Node A” and a second node 202b “Node B” coupled together and to various shared resources (e.g. memory, mass storage devices, I/O devices, etc.) via a shared interconnect including one or more scalability port switches. For example, Node A 202a of the illustrated embodiment may be coupled to and communicate with Node B 202b and various shared storage and I/O devices via a shared interconnect comprising a first scalability port switch (SPS) 212a “SPS-A” and a second SPS 212b “SPS-B”.

[0027] Node A 202a of the illustrated embodiment includes a plurality of processor packages 204a-d and a memory 206a. Each processor package 204a-d may contain one or more physical processor cores associated with one or more logical processors. In one embodiment, each of the illustrated processor packages 204a-d includes a pair of processor cores operable in an FRC mode as a single logical processor. Memory 206a is capable of storing data and/or processor-executable instructions and may comprise any suitable memory such as dynamic random access memory (DRAM) for

example. Node A 202a further includes a scalable node controller (SNC) 208a “SNC-A” coupled to each processor package 204a-d via a shared bus 205a and to memory 206a via a memory interconnect 210a. SNC-A 208a of the illustrated embodiment may be further utilized to coupled Node A 202a to SPS-A 212a and SPS-B 212b of the shared interconnect as illustrated. Additional nodes (e.g. Node B 202b) may include substantially identical components operating in a substantially identical manner as those of Node A 202a. Node B 202b of the illustrated embodiment has accordingly not been described in order to avoid unnecessarily obscuring the present invention.

[0028] Each SPS (212a, 212b) may in turn be coupled to a first I/O hub (IOH) 214a “IOH-A” and a second IOH 214b “IOH-B” as illustrated, allowing a greater number and variety of shared devices such as memory, mass storage devices (e.g. optical or magnetic storage devices), and I/O devices (e.g. display devices, keyboards, mice, trackballs, network interface cards, etc.) to be coupled to data processing system 200 via each IOH’s hub interfaces 222. For example, IOH-A 214a may be coupled to and communicate with a conventional I/O hub 216a “ICH2”, one or more peripheral component interconnect (PCI) or PCI extended (PCI-X) controller hubs 218a “P64H2”, and an Infiniband controller hub 220a “TVXB” via hub interfaces 222 as illustrated. IOH-B 214b may be similarly coupled to a conventional I/O hub 216b, PCI/PCI-X controller hubs 218a, and an Infiniband controller hub 220a as shown, however, in alternative embodiments of the present invention, each IOH may be coupled to any number or combination of controller hubs, mass storage, memory, or I/O devices, or the like.

[0029] ICH2 216a of the illustrated embodiment provides an interface to peripheral components or other devices for data processing system 200. ICH2 216a may comprise any suitable interface controller to provide for any suitable communication link to IOH-A 214a and/or to any suitable device or component in communication with ICH2 216a. ICH2 216a for one embodiment provides suitable buffering and arbitration for each interface. ICH2 216a may be further coupled to and provide an interface to a firmware hub (FWH) 224a as illustrated. FWH 224a may comprise any suitable interface controller to provide for any suitable communication link to ICH2 216a. In the illustrated embodiment, FWH 224a comprises a memory 226a to store one or more firmware interfaces (e.g. SAL or ACPI BIOS program code) according to

the present invention. Memory 226a may comprise any suitable memory capable of storing data and/or processor-executable instructions. In one embodiment, memory 226a comprises a non-volatile memory such as a read-only memory (ROM), programmable read-only memory (PROM), flash memory, or the like.

[0030] While a specific embodiment of data processing system 200 has been illustrated herein, it should be appreciated that additional alternative embodiments as well as variations or modifications of the illustrated exemplary embodiment may be implemented within the scope of the present invention. For example, in one alternative embodiment, data processing system 200 may be implemented with a single node having more or fewer components than the nodes 202a-b depicted. A node may accordingly be coupled to available resources without the use of a scalability port switch by coupling, for example an associated scalable node controller directly to one or more I/O hubs. Firmware interfaces (e.g. SAL or ACPI BIOS program code) according to the present invention may also be stored within Node A 202a either within existing memory 206a or an additional memory coupled to SNC-A 208a directly or via a conventional I/O hub and FWH as described with respect to memory 226a.

[0031] It should similarly be appreciated that the present invention may be practiced utilizing a data processing system 200 having a greater or lesser number of components as the illustrated exemplary system. For example, data processing system 200 may comprise, in alternative embodiments of the present invention, one of a wide variety of server or client computer systems or devices such as a workstation, personal computer, "thin client" (i.e. network computer or NetPC), Internet appliance, terminal, palmtop computing device, robust cellular or Personal Communications Services (PCS) telephone, "thin server" (sometimes called an appliance server, application server, or specialty server), or the like. In one embodiment, data processing system 200 comprises a high-availability server data processing system.

[0032] **Figure 3** illustrates an exemplary block diagram of processor package 204a of **Figure 2** according to one embodiment of the present invention. In the illustrated embodiment, a processor package 204a may be coupled to a shared "front side" bus 205a, and may include functional redundancy check (FRC) logic 308, a shared cache memory 306, and a first processor core 302a and a second processor core 302b, each having an associated private cache memory 304a and 304b, respectively. Within the

present description, the term “processor core” or “core” describes a physical processing element or execution entity, the term “processor package” or “package” describes a physical entity which incorporates, houses, or contains one or more processor cores. The term “processor” may therefore be utilized to describe either a physical processing element or execution entity, or a logical processing element or execution entity as viewed by system management or application software (e.g. operating system software).

[0033] In one embodiment, the first and second processor cores 302a-b within the depicted processor package 204a may be bound together to function as a single logical processor from the point of view of operating system software. Both cores 302a and 302b operate in an FRC mode in which identical instructions may be provided to each core and concurrently executed on identical data utilizing FRC logic 308. Processor cores 302a, 302b may each comprise any suitable processor architecture and for one embodiment comprise an Intel™ Architecture, used for example, in the Itanium™ family of processors available from Intel™ Corporation of Santa Clara, California. FRC logic 308 operates processor cores 302a-b as a single logical processor by concurrently providing each with identical data and instructions. Each processor core 302a-b executes instructions and processes data to generate results which may then be stored within an associated private (e.g. 304a-b) and/or shared (e.g. 306) cache. FRC logic 308 monitors processor core store operations or “writes” to shared cache 306 to detect inconsistencies in the generated results which indicate an FRC mismatch error. Such inconsistencies can occur as a result of partial or total hardware failure, or the effect of an alpha particle or other form of natural or artificial radiation impacting one or more processor cores.

[0034] In alternative embodiments, FRC logic 308 may be implemented as executable firmware or dedicated hardware coupled with or incorporated into one or more of the processor package’s 204a components. In yet another embodiment, a plurality of processor packages are provided, each including a single processor core with FRC logic being provided either within each processor package or external to each package to operate two or more of the package’s associated processor cores in an FRC mode.

[0035] **Figure 4a** illustrates a process flow diagram for a first portion of a method according to one embodiment of the present invention. The illustrated process begins

(block 402) and thereafter hardware (e.g. FRC logic) detects an error within a first processor core of a plurality of processor cores operating in an FRC mode as a single logical processor, disables the first processor core, and generates a system-transparent machine check exception (block 404) which in turn causes firmware (e.g. SAL program code) to generate a system-visible system control interrupt (SCI) to operating system (OS) software and return control of at least a second, error-free processor core of the plurality of processor cores to the OS (block 406). In alternative embodiments in which the described plurality includes three or more processor cores, additional error-free cores may also be turned over to operating system control.

[0036] OS software then resumes execution on the second processor core where it was interrupted by the machine check exception and queries firmware (e.g. ACPI BIOS) to determine the source of the SCI (block 408). In one embodiment, this may be accomplished by the OS executing an `_Lxx` control method. Firmware responds to the OS query by notifying the operating system of a request for processor hot removal (block 410). This may be accomplished using an ACPI Notify command. The operating system then modifies its internal data structures to indicate that the processor is offline and requests firmware ejection of the processor (block 412). In one embodiment, the operating system requests firmware ejection of the processor by executing an `_EJx` control method.

[0037] In response to the operating system ejection request, firmware virtually ejects the processor from the system, resets the processor, re-enables FRC operation of at least the first and second processor cores and generates an SCI to the Operating System (block 414) before beginning a second portion of the illustrated method embodiment depicted in **Figure 4b**. Thereafter, in the process illustrated in **Figure 4b**, operating system software queries firmware (e.g. ACPI BIOS) to determine the source of the previously-generated SCI (block 416). Firmware responds to the OS query by notifying the operating system of a request for processor hot insertion (block 418). Next, the operating system queries firmware to determine the status and to identify the associated resources of the processor to be inserted (block 420) and firmware provides the operating system with the requested data (block 422). In one embodiment, this may be accomplished by executing one or more control methods (e.g., `_STA`, `_MAT`, and `_CRS`). Once the relevant status and resource data is received, the operating system requests insertion of the processor (block 424) and the

firmware virtually inserts the processor into the system (block 426). In one embodiment, the operating system may request insertion of the processor using a _PS0 control method which puts a device into its D0 or “active” state. Once the processor has been inserted and/or activated as described, the OS modifies its internal data structures to indicate that the processor is online and available before the illustrated process terminates (block 430).

[0038] In the foregoing description, the present invention has been described with reference to specific exemplary embodiments thereof. It will be apparent however, that variations or modifications of the exemplary embodiments described as well as alternative embodiments of the present invention may be implemented without departing from the broader spirit or scope of the present invention as defined in the appended claims. The specification and drawings are accordingly to be regarded in an illustrative rather than a restrictive sense.